Universitat Politecnica de Catalunya

Mercè Álvarez de la Campa Crespo

3D virtual body from a single image

Supervised by:
Toni Susín

Master in Innovation and Research in Informatics
Facultat d'Informatica de Barcelona

April 2020

Contents

Chapter 1

Introduction

This thesis develops a fully automatic solution for computing a 3D textured human body avatar from a single image. There are many practical applications of single-view 3D avatar reconstruction. Mostly of the image data on Internet are captured from a single camera. And avatars are used in 3D printing, games, augmented and virtual reality (AR/VR). The ability to obtain 3D avatars from the real world can be very useful. And even more if the procedure is done in a single view instead of multiple views such as scans or videos that most of the pipelines uses, since it could reduce significantly the time and cost of the procedure.

Analysis of humans in 3D has many applications in different fields such as biology, psychology or neuroscience. More recent, there is a study on the psychology field of computer science where self look-alike 3D avatars are used in VR for a self-counselling method [18]. On this study, the participants were scanned in a previous session and then immersed in a self-conversation between their own 3D look-alike avatar and a Freud virtual avatar. They had to explain their problem to the virtual Freud and then do body-swapping and become Freud. Hence, each participant end up having a self-dialogue and giving advice to their selves as if they were seeing the problem from outside. This study shows that self-conversation method results in a better opportunity to help the participant and it can be useful as a strategy for self-counselling. In this study they scanned the full body of the participants and it took one session before the immersion in the VR scenario. Computing the 3D look-alike avatar from a single photo would reduce the time of the procedure.

There are other applications for look-alike avatars like virtual clothes fitting. This could be useful for clothes shops and a single view method would be simpler and cheaper to install on the store or even the customers could use the tool online in their homes with no need of complicated systems.

Chapter 2

State of the art

There are already methods that reconstruct 3D body meshes from 3D scans or videos and some that reconstructs this meshes from a single view picture. By combining the recent developments in automatic 2D joint detection [7] and 3D human modeling [14], we can achieve a 3D mesh of a human body from a single image [6]. The *SMPL* model they use for the method has several parameters that lets to change the pose and the shape of the initial body shape. Using a deep learning base algorithm, they introduce the parameters into the neural network in order to match the shape and the pose of the 3D mesh with the one in the picture (see Figure 2.1).

FIGURE 2.1: Results of the *SMPLify* method combining 2D joint detection and a 3D base model *SMPL* using deep learning.

There is another study on this line that they even achieve the facial expressions of the input image [15]. They compute 3D hands, face and body from a single image (see Figure 2.2) using an extended version of the *SMPL* model which has fully articulated hands and facial expressions called *SMPL-X*. The approach is similar to *SMPLify* method but with more input parameters for the neural network.

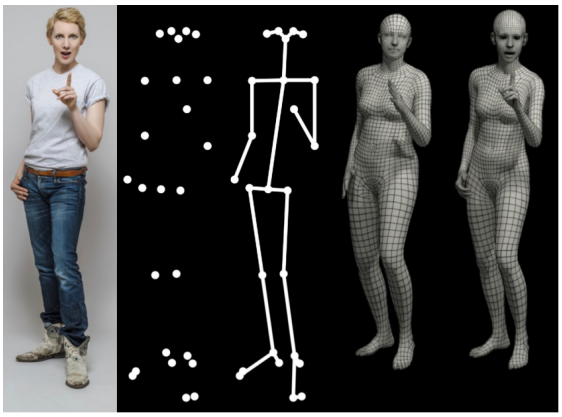


FIGURE 2.2: Results of the *SMPL eXpressive* using the extended model *SMPL-X* which contains articulated hands and facial expressions.

There is another method [3] that uses as well the *SMPL* base model but instead of combining the mesh with 2D joints, they use UV information [12] from the image to match the base mesh with the person (see Figure 2.3).

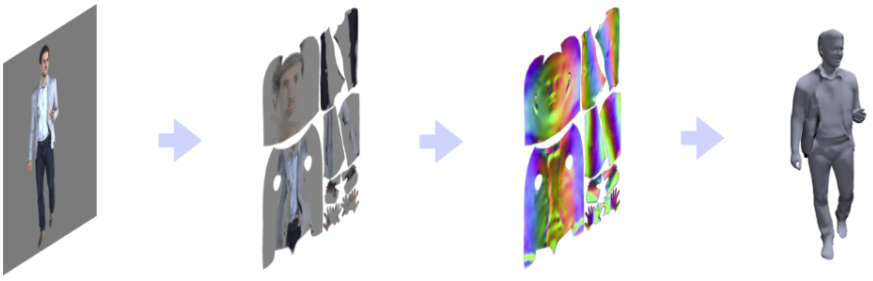


FIGURE 2.3: *Tex2Shape* method. Input image transformed into an incomplete texture, then the method converts the texture into normal and displacement maps. The maps give detail to the final mesh.

This three methods mentioned [3, 6, 15] reconstructs a 3D mesh from a single view with good accuracy. But they just provide the mesh and not a final UV texture. In the last method [3] they only provide a partial UV texture from the visible view of the image.

There is another approach that computes a final 3D avatar with textures [2] using a single face photo called *Headsot* (see Figure 2.4). But it only takes the face of the person and the resulting avatar has a face similar to the person in the image but there is no estimation on the body. All the details of the avatar (hair, body shape, clothes) are generated manually with the program. Although it does only estimates the face of the person, this method can be very useful depending the application is used for. For example if we only want the face info in a game or movie but then change as we want the clothes manually, this can save time in the graphic designer side.

FIGURE 2.4: *Headshot* program that creates an avatar from a single image.

We can find more methods that generates 3D faces from a single photo with high quality results. Such as *PRNet* [11] that generates a 3D mesh face with texture (see Figure 2.5) or the program *AvatarSDK* [1] that from a single photo generates as well hair-styles (see Figure 2.6).

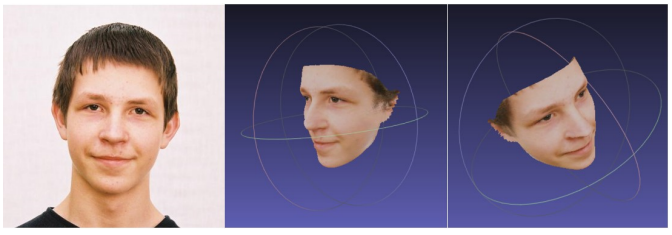


FIGURE 2.5: *PRNet* is a method that computes 3D joint face points and reconstruct a 3D face shape from them.

FIGURE 2.6: *AvatarSDK* application that creates head avatars from a single image with a fixed topology and a reconstructed person's hairstyle.

This thesis will focus the study on achieve a final 3D full body textured avatar ready for using on different applications such as virtual reality environments or games. We will try to obtain an avatar with the shape and texture as the person in the image from a single view and we will achieve this procedure automatically.

Chapter 3

Method explained

We are interested on a simple method to extract a 3D avatar from a single RGB image. We need to extract all the information from a single picture to obtain a final 3D textured avatar capable to use it for external applications.

From 2D RGB information we are going to obtain a 3D mesh close to the shape of the person, this mesh will be rigged and we will obtain finally a UV texture attached to the mesh with the color information. On the way we will find some artifacts since we are going from 2D information in a single view direction to 3D information. There will be occluded data but we are going to try to come up with some solutions to extract all the information needed to compute the final 3D rigged avatar.

3.1 Pose and face keypoints estimation

First of all, we want to compute human pose and face keypoints of the person in the input image. There are several methods to accomplish this step but we are going to use *OpenPose* because is one approach with the best results considering the trade-off between speed and accuracy (see Figure 3.1).

Most of this framework is based on [7]. And the face keypoint detectors are a combination of [7] and [17].

OpenPose

A realtime approach to detect 2D pose of multiple people in an image. They use Part Affinity Fields to associate body parts with individuals in the image, (see Figure 3.2 and 3.3).

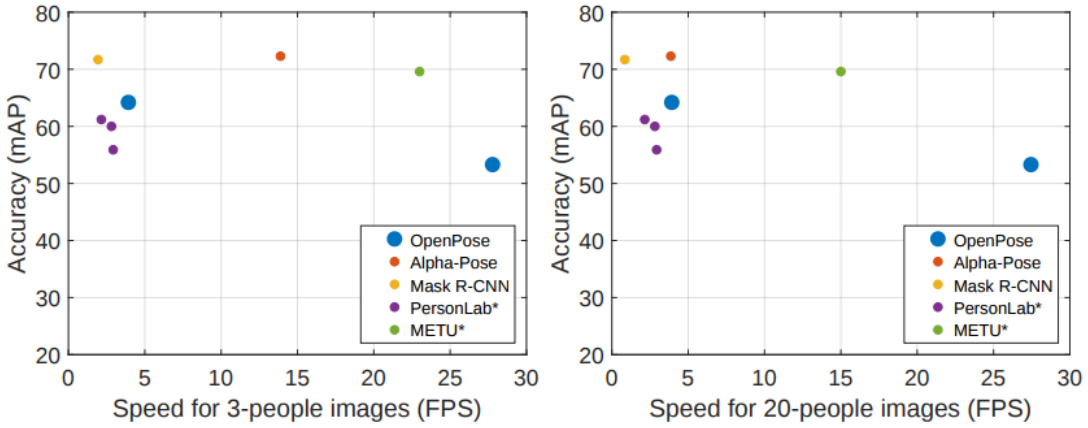


FIGURE 3.1: Trade-off between speed and accuracy for the main approaches to compute human pose keypoints. Algorithms with several values in the graph represent different resolution configurations. AlphaPose, METU, OpenPose provide the best results considering the trade-off between speed and accuracy. and accuracy.

FIGURE 3.2: **Top:** Multi-person pose estimation in a single image. **Bottom:** Part Affinity Fields corresponding to the limb connecting right elbow and wrist. The orientation is encoded with color.

Openpose consists on three parts: (a) body keypoints detection, (b) hand keypoints detection and (c) face keypoint detection. The core block is the first one and it was trained with three different datasets. These dataset collect images in diverse scenarios that contain many real-world challendes such as crowding, scale variation, occlusion, and contact. And all of this images are labelled, for example they have annotations on human body joints.

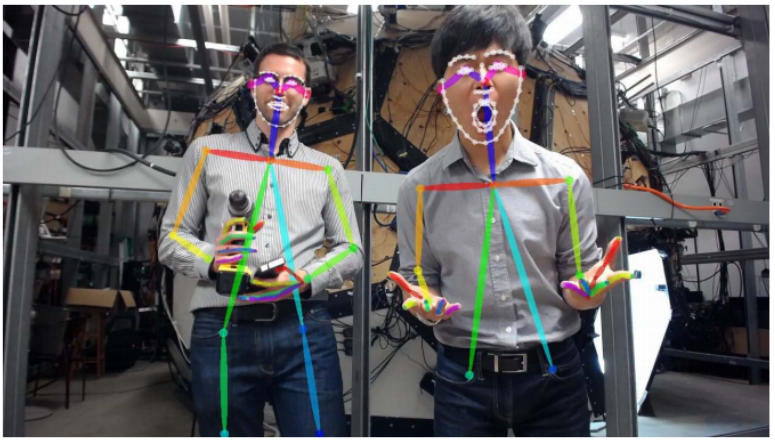


FIGURE 3.3: OpenPose detects body, hand and facial keypoints.

- *MPII Human Pose dataset [4]*: Includes around 25K images containing over 40K people with annotated body joints (14 body parts, see Figure 3.4a). The images were systematically collected using an established taxonomy of every day human activities. Overall the dataset covers 410 human activities and each image is provided with an activity label. Each image was extracted from a YouTube video and provided with preceding and following un-annotated frames. In addition, for the test set they obtained richer annotations including body part occlusions and 3D torso and head orientations.

- *COCO dataset [13]:* Includes more than 330k images containing 250k people with annotated 17 keypoints (12 human body parts and 5 facial keypoints, see Figure 3.4b). A part from [4], it extends to non-human information. It covers 80 object and 91 stuff categories. COCO is a large-scale object detection, segmentation, and captioning dataset.

- *COCO+Foot dataset:* It uses the COCO dataset and a small subset of foot instances out of the COCO dataset with 15k annotations (6 foot keypoints are labeled, see Figure 3.4c).

They extract the facial and hand keypoints from some of the output body keypoints. The algorithm used to extract the hand points [17] is the same they used for estimating the face keypoints.

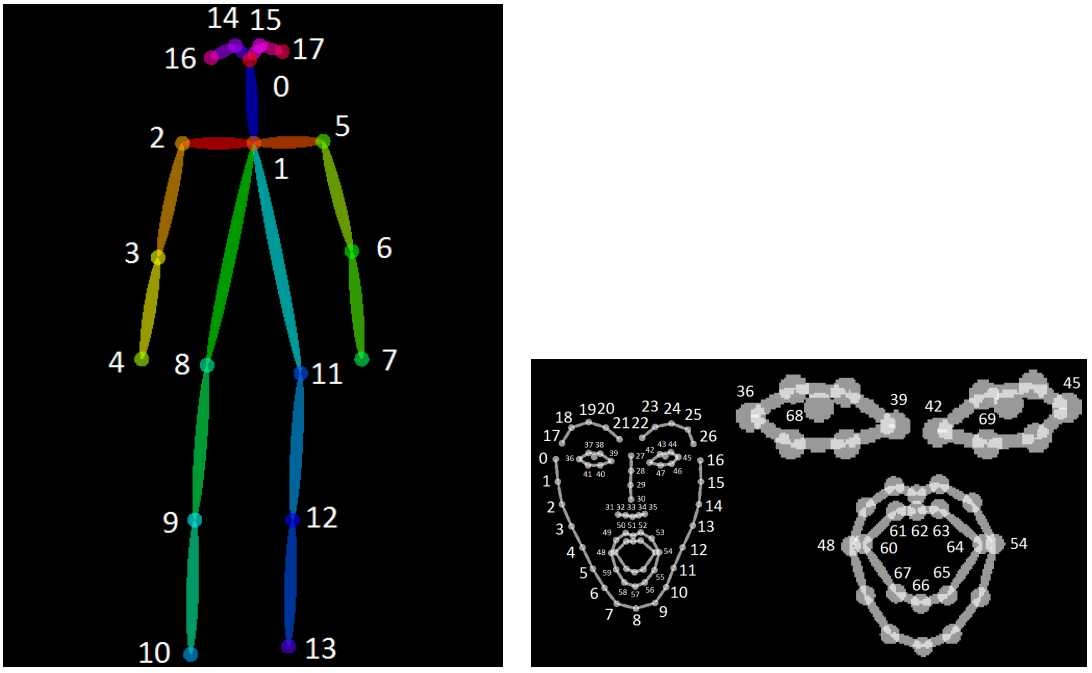
Model	CUDA	CPU-only
Original MPII model	73 ms	2309 ms
Original COCO model	74 ms	2407 ms
Body+foot model	36 ms	10396 ms

TABLE 3.1: Runtime difference between the 3 different body detection used in OpenPose with CUDA and CPU-only versions. They used a NVIDIA GeForce GTX-1080 Ti GPU and a i7-6850K CPU.

(A) MPII (B) COCO (C) COCO+Foot

FIGURE 3.4: Keypoint annotation configuration for the 3 datasets used in OpenPose.

For body keypoints estimation with GPU, OpenPose is faster when uses the Body+foot model in comparison with the original MPII and COCO model (see Table 3.1). But when we use the CPU-only instead, the runtime increases compared to the other two methods. In the CUDA version, the difference in runtime is minor than the CPU-only case. For that, we are going to use for our method one of the two original models. As the **COCO model** is based on a dataset with more images and information, we are going to use finally this model for our pipeline. We are going to discart hand keypoints estimation to reduce the complexity and the overall runtime.



(A) Pose Output Format (COCO model) (B) Face Output Format

FIGURE 3.5: Openpose Output for our pipeline

The OpenPose output (see Figure 3.5) will be stored in a JSON file with the pose and face keypoints in texture coordinates (and a third confident value).


```
{
    "version":1.1,
    "people":[
        {
            "pose_keypoints_2d":[582.349,507.866,0.845918,746.975,631.307,0.587007,...],
            "face_keypoints_2d":[468.725,715.636,0.189116,554.963,652.863,0.665039,...],
            "hand_left_keypoints_2d":[746.975,631.307,0.587007,615.659,617.567,0.377899,...],
            "hand_right_keypoints_2d":[617.581,472.65,0.797508,0,0,0,723.431,462.783,0.88765,...]
            "pose_keypoints_3d":[582.349,507.866,507.866,0.845918,507.866,746.975,631.307,0.587007,...],
            "face_keypoints_3d":[468.725,715.636,715.636,0.189116,715.636,554.963,652.863,0.665039,...],
            "hand_left_keypoints_3d":[746.975,631.307,631.307,0.587007,631.307,615.659,617.567,0.377899,...],
            "hand_right_keypoints_3d":[617.581,472.65,472.65,0.797508,472.65,0,0,0,723.431,462.783,0.88765,...]
        }
    ],
    // If `--part_candidates` enabled
    "part_candidates":[
        {
            "0":[296.994,258.976,0.845918,238.996,365.027,0.189116],
            "1":[381.024,321.984,0.587007],
            "2":[313.996,314.97,0.377899],
            "3":[238.996,365.027,0.189116],
            "4":[283.015,332.986,0.665039],
            "5":[457.987,324.003,0.430488,283.015,332.986,0.665039],
            "6":[],
            "7":[],
            "8":[],
            "9":[],
            "10":[],
            "11":[],
            "12":[],
            "13":[],
            "14":[293.001,242.991,0.674305],
            "15":[314.978,241,0.797508],
            "16":[],
            "17":[369.007,235.964,0.88765]
        }
    ]
}
```


FIGURE 3.6: OpenPose output stored in a JSON file.

The framework stores different information (see Figure 3.6), but we are going to consider only the **pose_keypoints_2d** and **face_keypoints_2d** for our pipeline.

3.2 Silhouette estimation

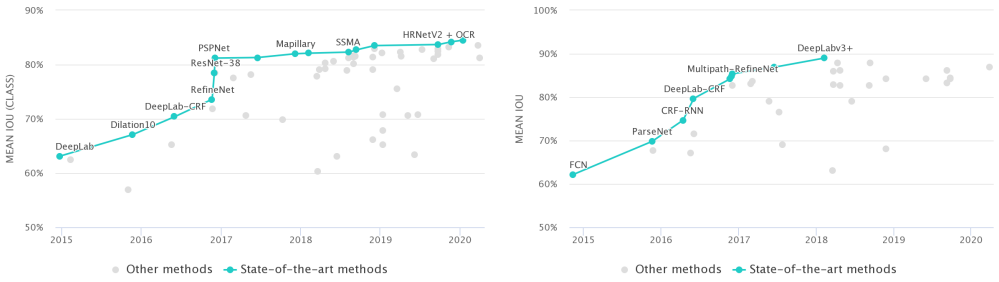
A part from extracting the pose and face keypoints we want to estimate as well the silhouette of the person. This will be useful to extract the part of the image that corresponds to the person and avoid other details and data of the image that does not belongs to the person. The silhouette, in combination with the keypoints, will be useful to compute the 2d human shape in the image. This will be used to extract a 3D mesh that matches more to the shape in 2D.

Semantic segmentation, or image segmentation, is the task of clustering parts of an image together which belong to the same object class. It is a form of pixel-level prediction because each pixel in an image is classified according to a category. And there are some algorithms that are able to extract people silhouettes.

Most of the semantic segmentation algorithms are trained with the Cityscapes, PASCAL VOC or ADE20K dataset as benchmark.

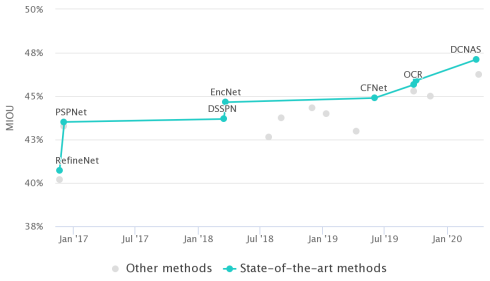
The Cityscapes Dataset focuses on semantic understanding of urban street scenes. It contains over 5k annotated images with fine annotations and 30 classes (including person). The PASCAL VOC dataset contains over 10k annotated images with 20 classes and 3k segmented images (1k containing at least one segmented person). And finally, ADE20K dataset contains over 20k labelled images and 150 classes (about 10k images containing at leas one segmented person or parts of it).

Since this datasets contains the category of person labelled, all are useful as reference for a method that is able to detect persons and its silhouette. Models are usually evaluated with the Mean Intersection-Over-Union (Mean IoU) and Pixel Accuracy metrics. After comparing some methods (see Figure 3.7), we finally select *DeepLabv3+* algorithm that uses PASCAL VOC dataset.



(A) Semantic segmentation on Cityscapes test dataset

(B) Semantic segmentation on PASCAL VOC 2012 test



(C) Semantic segmentation on ADE20K val dataset

FIGURE 3.7: Semantic segmentation using different models and datasets.

DeepLab

A state-of-art deep learning model for semantic image segmentation [8],[9], where the goal is to assign semantic labels (e.g., person, dog, cat and so on) to every pixel in the input image (see Figure 3.8).

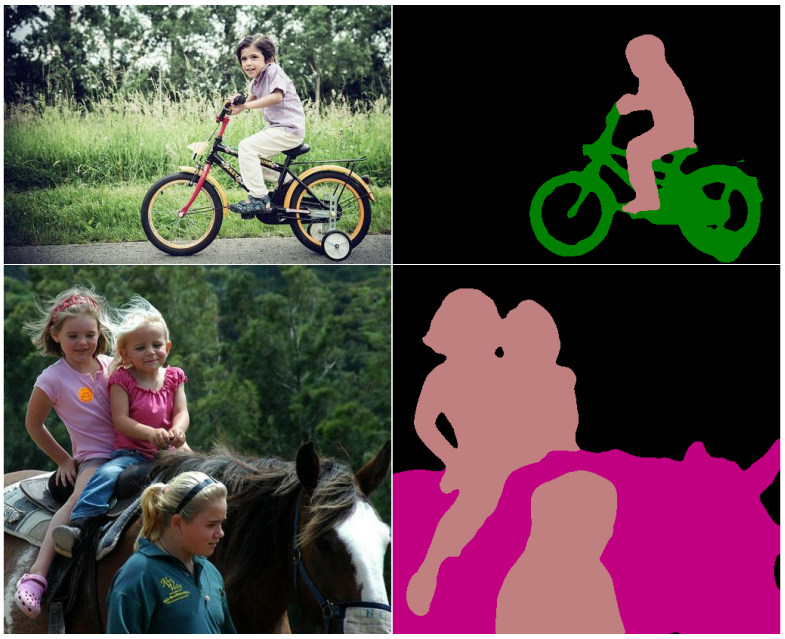


FIGURE 3.8: DeepLab results from an input image.

These images are segmented using a color indexed. Each color index represents a unique class (with unique color) known as a color map (see Figure 3.9).

FIGURE 3.9: PASCAL VOC label color map.

For our pipeline we are interested on detecting the person in the image and this is indexed as pink color in the output segmentation image (see Figure 3.10). Other categories will be ignored.

FIGURE 3.10: Input image (left) and silhouette (right).

3.3 SMPL mesh model

Once we obtain from the input image the keypoints and the silhouette, combining both outputs we can extract more detailed 2D information of the person, such as body parts, size and shape of each part in 2D. All of this will be useful to extract the 3D mesh of the person in the image.

We are going to use a Skinned Multi-Person Linear Model (*SMPL*) [14] mesh model as a template mesh. It is a realistic 3D model of the human body that is based on skinning and blend shapes and is learned from thousands of 3D body scans (see Figure 3.11 and Figure 3.12).

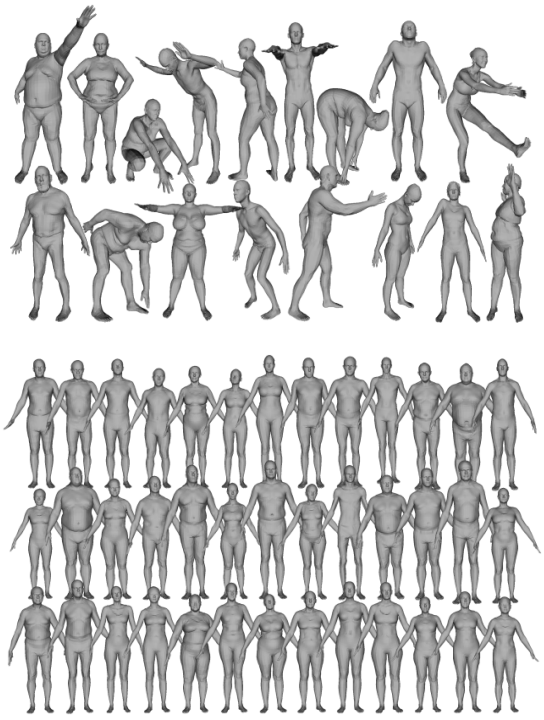


FIGURE 3.11: 3D body scans with different poses and shapes.

FIGURE 3.12: SMPL model (orange) fit to 3D scanned meshes (gray).

The base model contains pose θ and shape β parameters that lets to deform the mesh. To be more precise, it has 207 pose blend shapes (23 joints x 9, each bone has a 3x3 rotation matrix) that allows to deform the mesh of the model in order to get a specific pose (the same a skeleton does in a rigged mesh but using bone weights instead of blend shapes). And an additional 10 body shape parameters to deform in different ways the shape of the mesh. For example, the first parameter (or principal component) mostly changes the size or height of the mesh and the second parameter mostly the belly part.

The authors of SMPL provide the model in PKL file format (separate male and female model files) and code functions in python to load and save SMPL models and a sample script. We are going to use that to generate a template database of different body shapes and then we will fit the one that matches best to the shape of the person in the image. For that, we load the SMPL model in PKL file format, then we set different values of β parameter and then we save each body as a mesh in .OBJ format (see Code 3.1).

```
from smpl_webuser.serialization import load_model
import numpy as np
import math

## Load SMPL model (here we load the female model)
m = load_model( './models/basicModel_f_lbs_10_207_0_v1.0.0.pkl' )

## We vary every component of the betas from -5 to 5
for beta in range (0,10):
    for i in range(-5,5+1):
        value = i/1.0
        m.betas[beta] = value
        ## Write to an .obj file
        outmesh_path = './templates_female/template_female_smpl_pose_beta_'+str(beta)+'_'+str(value)+'.obj'
        with open( outmesh_path, 'w') as fp:
            for v in m.r:
                fp.write( 'v %f %f %f\n' % ( v[0], v[1], v[2]) )
            for f in m.f+1: # Faces are 1-based, not 0-based in obj files
                fp.write( 'f %d %d %d\n' %  (f[0], f[1], f[2]) )
        m.betas[:] = 0
```

LISTING 3.1: Python code to generate SMPL models (female case) with different shape bodies and saved them as .OBJ format.

Instead of generating a dataset of template bodies with different shapes to after select the one that fits best to the 2D person shape of the input image, we could use the β parameters directly into an optimization function and compute the values for each β parameter that minimizes that function [6], but we wanted to make things simpler since our goal is to come up with a pipeline that obtains a 3d rigged avatar from an input image in a simple way but enough quality results.

They provide as well the SMPL model in animated .FBX format with UV textures (see Figure 3.13).

FIGURE 3.13: Animated male model (Left) and the UV texture of the SMPL model (Right).

3.4 Segmentation and Optimization

In order to select the mesh that fits best to the person in the image, we are going to consider the simple case where the person is photographed from the front view. Following the human standard anatomy [10], in the image we are going to segment the body taking into account the head size and we focus on the belly part (see Figure 3.14) to compare with the data base of templates we generated. We did previously the same partition in each mesh of the data base.

For each of the sections we extract a deterministic number of equally spatial points in terms of head distances. We focus on vertices in the surface (silhouette) and in the belly section. We extracted previously the same information from the meshes in the data set of SMPL templates with different body shape and saved it in a matrix (see Figure 3.15 and 3.16). The matrix has a dimension of 10 (different β parameters) x 11 (each β parameter has values from -5 to 5 with a value step of 1), inside each cell we have the 10 value points saved.

After we extracted the distance values in the input image, we compare this values to the matrix. Each cell of the matrix corresponds to a certain template body shape and we compute the mean Euclidean distances between each corresponding pair of values between the template and the image. We save this new values into a correlation matrix and the smallest value will give us the mesh of the data set that fits best to the belly part of the person in the image (see Figure 3.17). This process is done in *Matlab*.

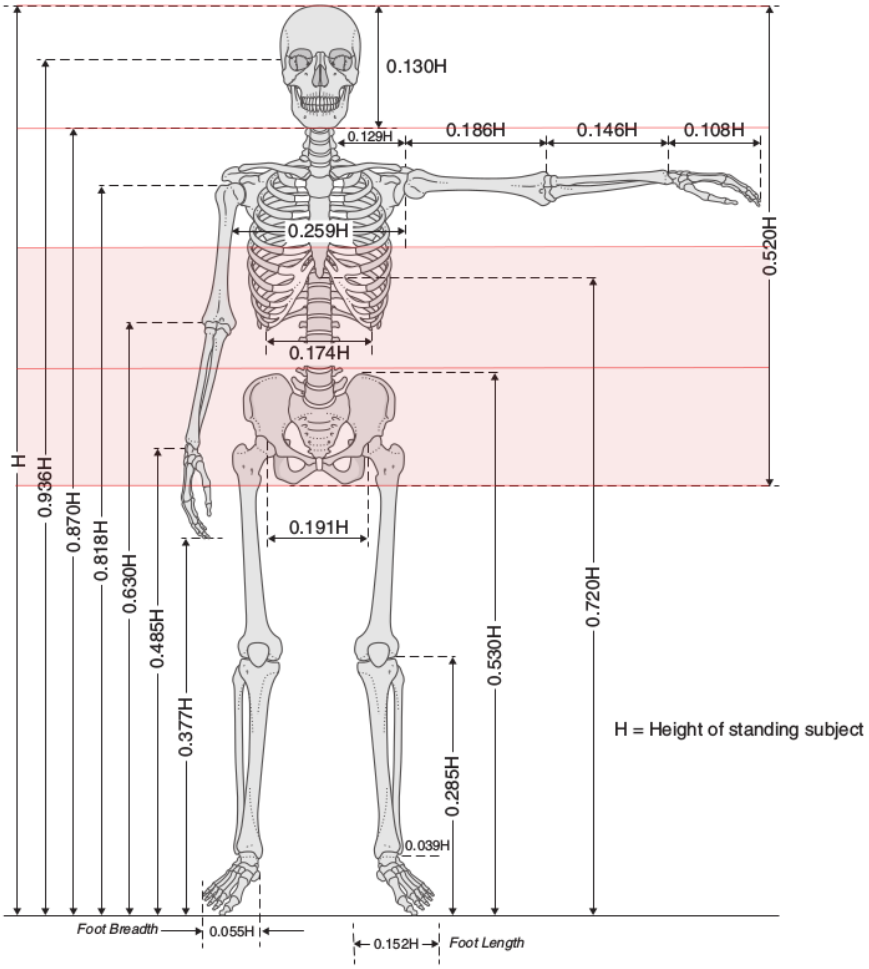


FIGURE 3.14: Segment length expressed as ratio of body height H (image modified from [16]). We divide the body by the head size. The area we are going to compare in red color.

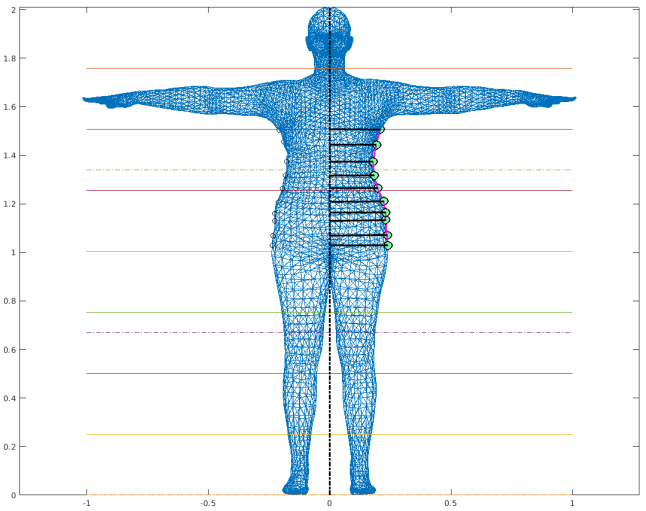


FIGURE 3.15: Segmentation of a mesh from the template data set.

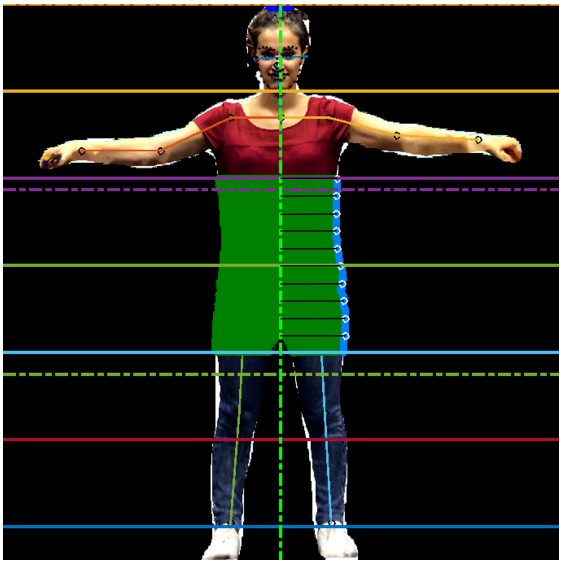


FIGURE 3.16: Segmentation of the person in the image.

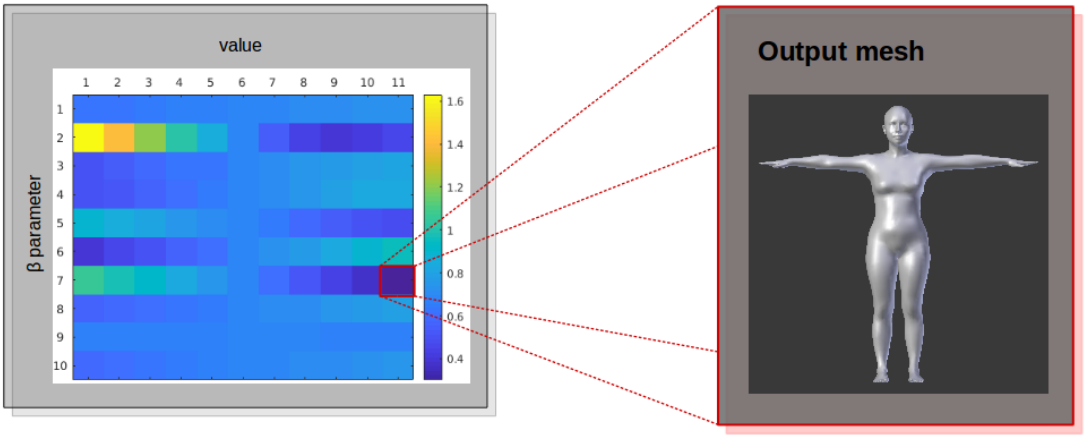


FIGURE 3.17: We compute the nearest matching template shape through an optimization process. Using a correlation matrix, we determine the closest template shape to the image person.

3.5 Texture projection

Once we have the mesh that fits best to the person in the image, we want to project the image onto the body object to obtain a textured avatar as a result.

We start with two reference points - one for the source of the projection L and another for the centre of the image plane O.

For each sample point on the surface we need to determine the vector of the ray from the projection source, through the image plane and to the surface. This can be achieved by subtracting the vector representing the source of the projection from the point on the surface.

To determine the point in the image texture that would be projected onto any particular point on the surface we need to be able to measure how far the ray travels in specific directions - in the Y direction of the image plane, in the X direction of the image plane, and also the distance travelled perpendicular to the image plane.

The vector between the two reference points L and O can be used as the *centreline* of the projection and this will be perpendicular to the image plane and as well perpendicular to the plane where the ray intersects the mesh. It can be calculated simply by subtracting one point from the other ($\vec{LO}$).

The plane where the ray intersects the mesh has the point Q as the center. The $\vec{X}$ and $\vec{Y}$ vectors need to both be perpendicular to the *centreline* and to each other - to ensure they are independent (see Figure 3.18).

According to the *Thales's theorem*, two triangles are similar when they have equal angles and proportional sides. We can compute the vector $\vec{t} = (u, v)$ from the relative distances between the source L, the projection plane with center O and the plane at intersection of the mesh with center Q (see Figure 3.19).

$$\frac{\overline{vO}}{\overline{LO}} = \frac{\overline{YQ}}{\overline{LQ}} \longrightarrow \overline{vO} = \overline{v - (0,0)} = v = \frac{\overline{YQ} \cdot \overline{LO}}{\overline{LQ}} \tag{3.1}$$

$$\frac{\overline{uO}}{\overline{LO}} = \frac{\overline{XQ}}{\overline{LQ}} \longrightarrow \overline{uO} = \overline{u - (0,0)} = u = \frac{\overline{XQ} \cdot \overline{LO}}{\overline{LQ}} \tag{3.2}$$

Then,

$$\vec{t} = (u, v) = (\frac{\overline{XQ} \cdot \overline{LO}}{\overline{LQ}}, \frac{\overline{YQ} \cdot \overline{LO}}{\overline{LQ}}) \tag{3.3}$$

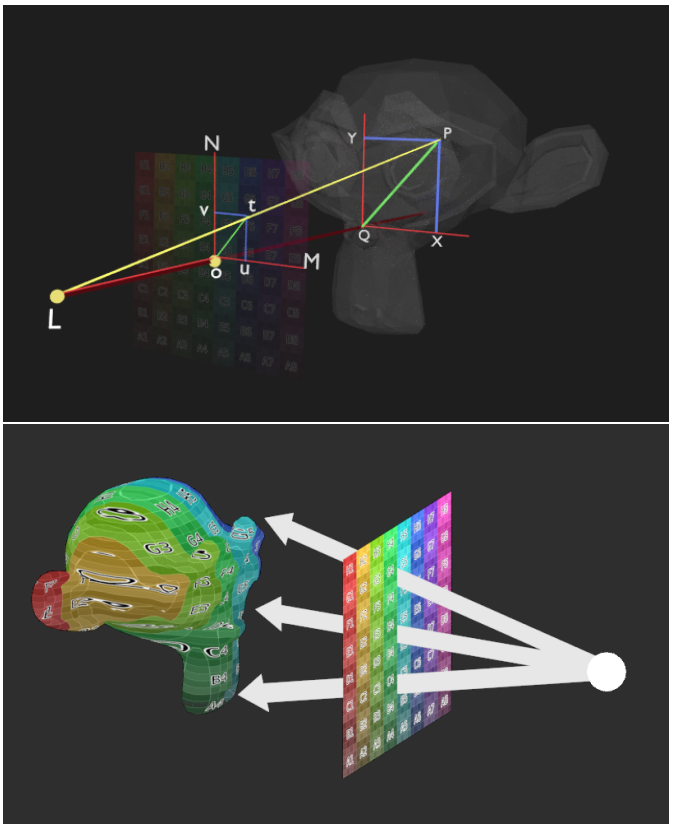


FIGURE 3.18: Image projection onto a mesh.

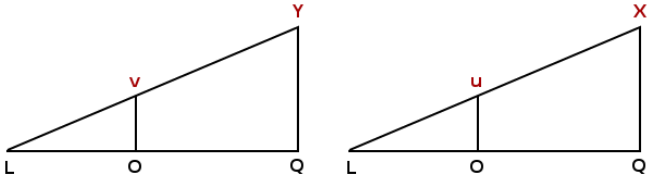


FIGURE 3.19: Similar triangles in the plane perpendicular to $\vec{M}$ (left) and $\vec{N}$ (right).

The computed coordinates of $\vec{t} = (u, v)$ in the image plane for a given surface point P (Eq. 3.3) will provide us the color that will be projected on this point of the surface mesh.

We compute the projection of the image onto the mesh using *Blender*. We attach the input image into a plane perpendicular to the avatar. After, we add to the avatar mesh a *UVProject* modifier where we set the plane that contains the input image as the projector and we use the original UV Map of the avatar.

Once we have the image projected onto the mesh, before baking the image onto the body avatar, we have to do a prior step because the person in the image is in another pose than the default pose of the mesh (T-pose) (see Figure 3.20).

Fitting avatar pose to image

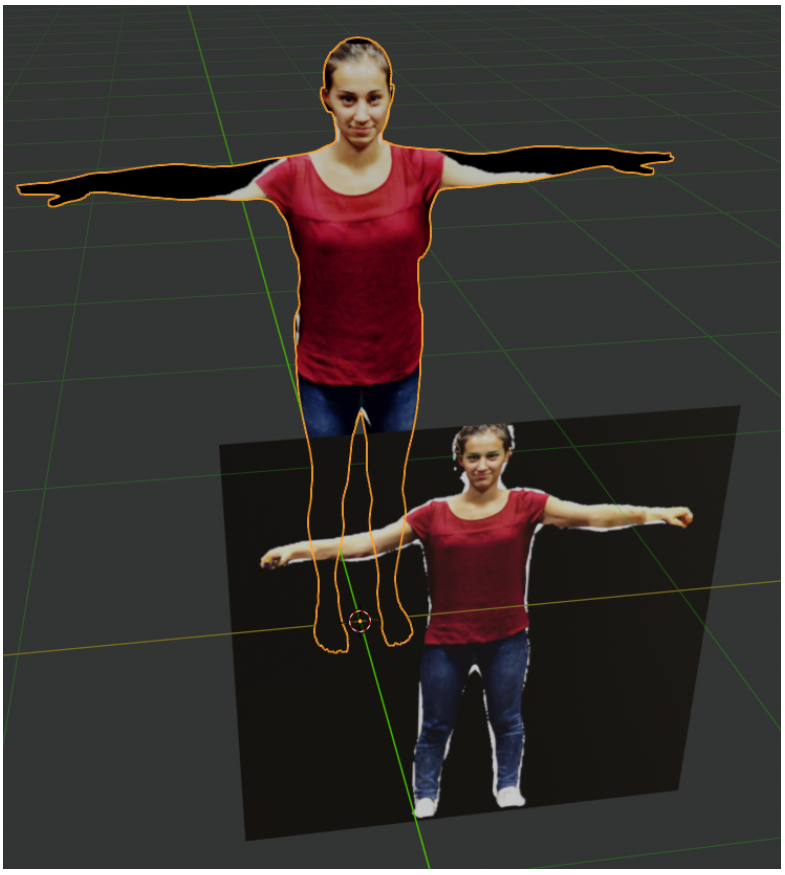


FIGURE 3.20: Input image projected onto the mesh.

Since the authors of the SMPL model provides as well them in animated .FBX format, we have the skeleton and the bone weights in the base mesh (see Figure 3.13). We are going to adjust the pose of the base mesh to the pose of the person in the image.

The pose of the person in the image that we computed from *OpenPose* is in image space and the skeleton pose of the base mesh is in world space (see Figure 3.21). In order to compare both poses we need to work in the same coordinate system.

First of all, we convert the *OpenPose* key points form the image space into world space (see Figure 3.22).

Once we have the skeleton bone position of the avatar and the pose key points both in world space (see Figure 3.23), we are able to compare both poses. We compute a vector form each pair of consecutive points of the avatar pose and the pose key points and then we extract how much the pose of the avatar differs respect to the *OpenPose* in term of angles (α). We rotate each bone of the avatar with the angle computed. (see Figure 3.24).

After matching the skeleton pose to the person's image pose (see Figure 3.25), we project and bake the image onto the mesh in order to obtain the UV texture (see Figure

FIGURE 3.21: Input image with the pose plotted in image space (left) and avatar in t-pose in world space (right).

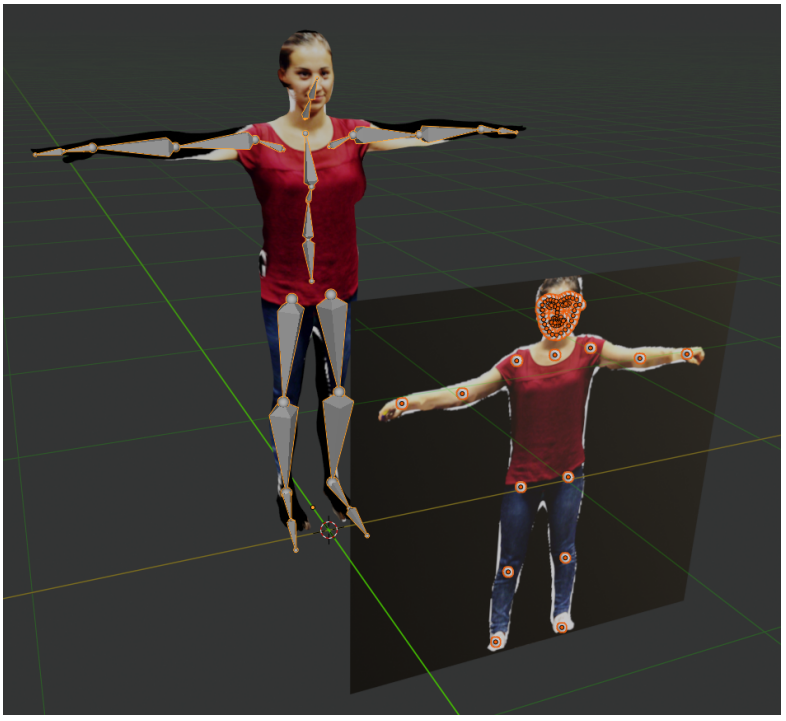


FIGURE 3.22: Avatar skeleton and pose key points of the initial image both in world space.

3.26).

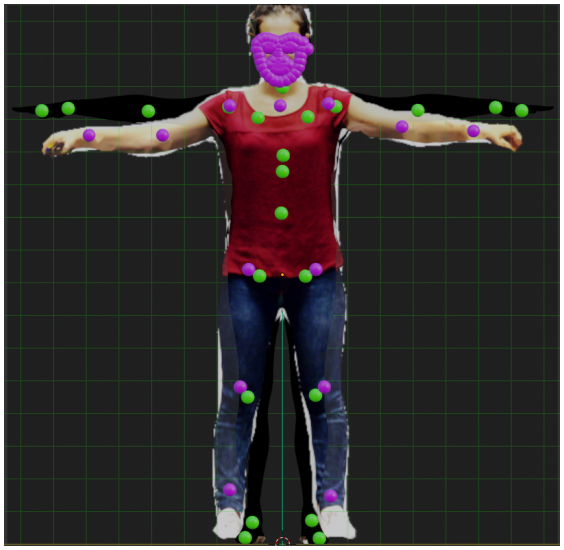


FIGURE 3.23: Avatar skeleton bone positions (green) and pose key points of the initial image (purple) both in world space.

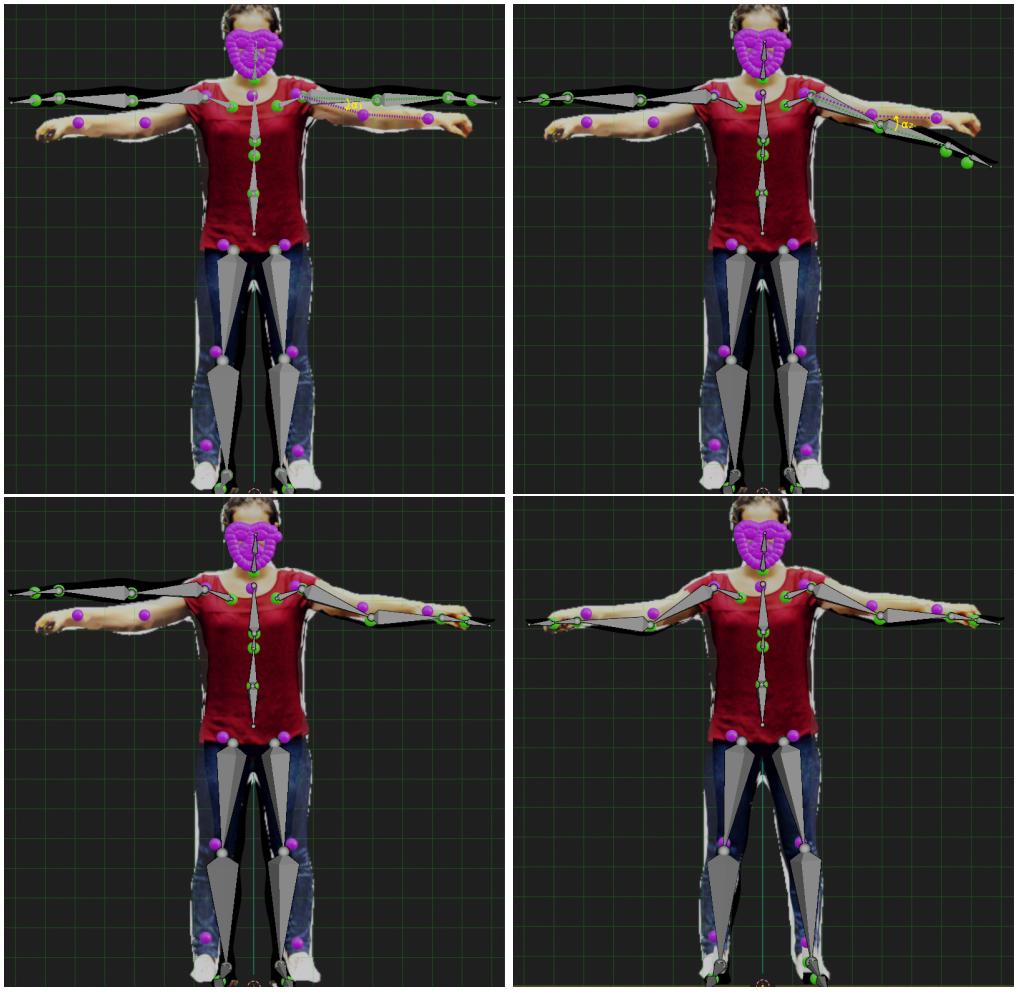


FIGURE 3.24: From each pair of points, we compute the angles α_i between the avatar skeleton pose and the image pose. And we rotate the corresponding skeleton bone with this angle.

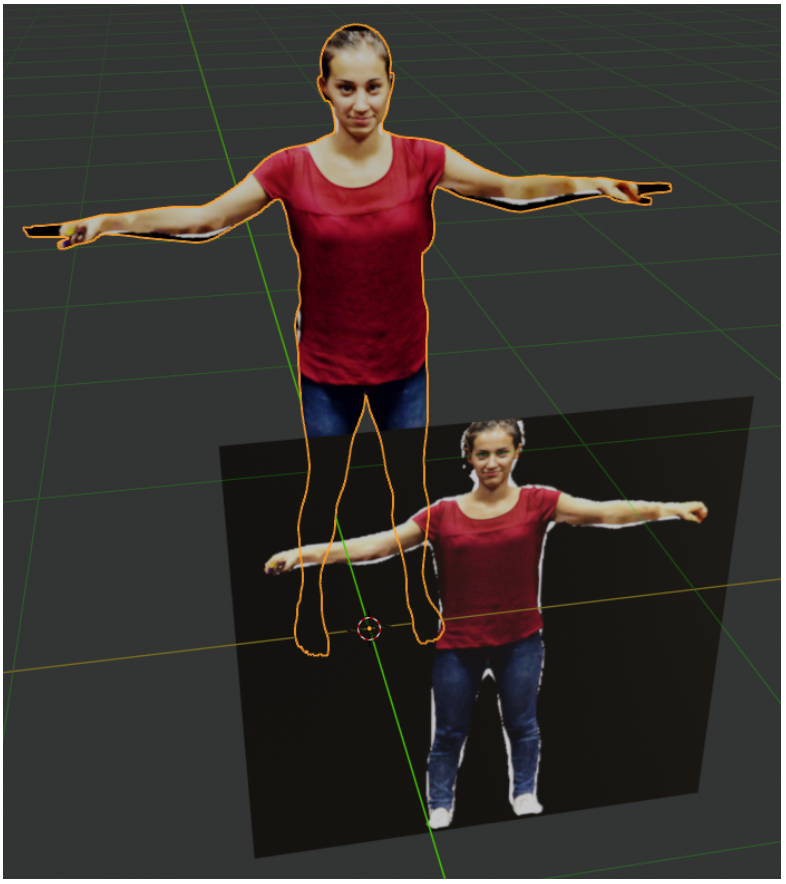


FIGURE 3.25: Pose mesh matches with person pose.

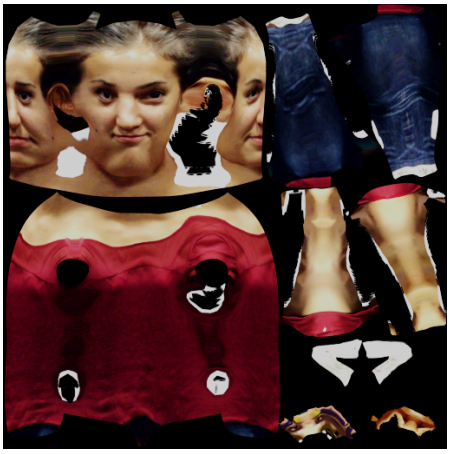


FIGURE 3.26: The resulting UV baked texture.

3.6 Texture correction

If we compare with the initial UV texture of the model, the texture we computed has black areas because there is missing information since we are limited to one view direction (frontal view). Regarding the back vertices of the mesh, they are intersected as well from the projection, resulting on the input image projected both in the front and back of the body mesh. This solve part of the problem of missing data but it introduce another errors like a duplicated face.

Since the UV map is the same for all the meshes because the topology does not change, we are going to use different masks to fix the areas of the texture we want.

In order to work with the images we are going to use the package of *OpenCV (cv2)* in *Python*.

Filling black/empy areas

For this part, we are going to fill the black areas of the image using the color of the region neighborhood. In other words, we are going to replace those bad pixels with its neighbouring pixels so that it looks like the neighbourhood.

OpenCV library provides two algorithms [5, 19] for archiving this task and both can be accessed by the same function *cv2.inpaint()*:

$$dst = cv2.inpaint(src, inpaintMask, inpaintRadius, flags[, dst]) \tag{3.4}$$

Where *src* is the input image (8-bit, 16-bit unsigned or 32-bit float 1-channel or 8-bit 3-channel image), *inpaintMask* is the mask that contains non-zero pixels indicating the area that needs to be inpainted (8-bit 1-channel image), *dst* is the output image with the same size and type as *src*, *inpaintRadius* is the radius of a circular neighborhood of each point inpainted that is considered by the algorithm and *flags* is the inpainting method that could be cv2.INPAINT_NS which use the Navier-Stokes based method [5] or cv2.INPAINT_TELEA that uses the algorithm proposed by Alexandru Telea [19].

We need to create a mask of same size as the input image, where non-zero pixels corresponds to the area which is to be inpainted. The initial mask takes the missing areas of the texture (the black areas) but there are some pixels with wrong colors (mostly the ones near the silhouette of the person), so we are going to modify the initial mask in order to take into account more pixels from the neighbor. In order to achieve that, we blur the first mask by convolving the image with a low-pass filter kernel. After applying this filter, the edges of the black areas are blurred. The are different types of blurring techniques but we use the averaging one. This is done using a normalized box filter.

It simply takes the average of all the pixels under kernel area and replaces the central element with this average. This is done by the function *cv2.blur()* and we specify the width and height of the kernel. A 3x3 normalized box filter would look like this:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{3.5}$$

In our case we apply a kernel of 15x15 size onto the image mask. And then, we create another mask taking into account only the black pixels. The resulting image mask has bigger areas than the original mask (see Figure 3.27). Then we apply the mask onto the image in order to get a texture with bigger black areas (see Figure 3.28) to remove some bad colors in the frontiers.

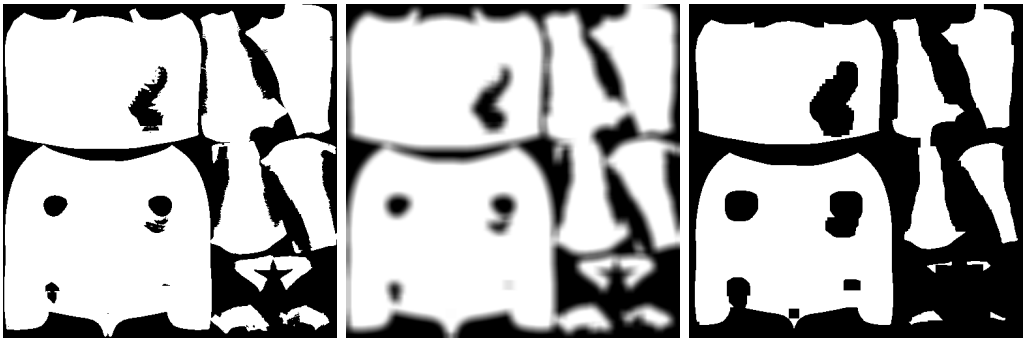


FIGURE 3.27: Initial mask taking into account the black areas (left), blurred mask using a kernel of 15x15 size (middle) and resulting mask after applying the filter (right).

FIGURE 3.28: Original baked image (left) and baked image with amplified black areas compared to the original (right).

Once we have the mask ready, we apply the inpaint function to compute the color of the black pixels by extrapolating the colors from the edges of the black areas. Both algorithms for inpainting gives similar results, we are going to use for our pipeline the one proposed by Alexandru Telea [19] (see Figure 3.29).

FIGURE 3.29: Initial baked image with amplified missing (black) areas (top left), mask (top right). resulting image of the Navier-Stokes based algorithm (bottom left) and the resulting image of the algorithm proposed by Alexandru Telea (bottom right).

Remove duplicated face

In order to remove the duplicated face in the texture, we are going to follow the next two steps:

1. Compute approximation hair color.
2. Apply the color computed in step 1 using a mask which considers the hair area in the image.

1. APPROXIMATE HAIR COLOR

In order to compute the hair color of the avatar, we first select a correct colored hair area in the texture. This area will be fixed for all the input images since the topology of the mesh does not change. And the color of this area will give us the color of the hair. This procedure is the same as in the section of filling black/empty areas (see Figure 3.29). But in this case, we will have as input of the algorithm a colored image only in the area we extract the hair color. The mask will consider the rest of the image and the output image will extrapolate the color pixels that are inside the mask area (see Figure 3.30).

2. APPLY HAIR COLOR TO THE UV IMAGE

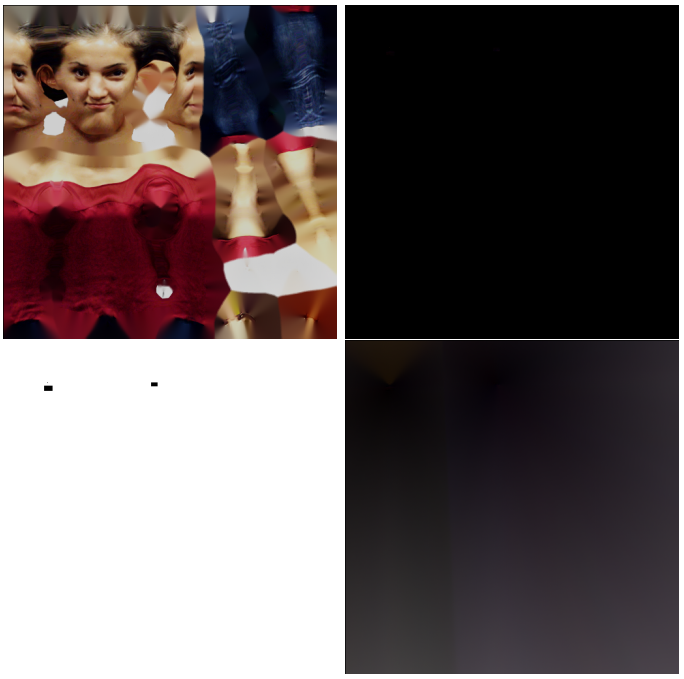


FIGURE 3.30: Initial baked image with empty areas filled (top left), input image for the inpaint algorithm which considers a hair section (top right). mask which selects the rest of the image pixels (bottom left) and resulting image from the algorithm (bottom right).

We blur the borders of the painted area in order to change gradually the color in the borders of the area with the rest of the image. Once we have the approximate hair color computed, we apply it combining the original image texture and the hair color image using a mask that takes into account the area of the hair. As the topology of the mesh does not change for each avatar, we can create a unique mask for all input images, same as the previous step. We can do the mask manually with an image editor program like *GIMP*. Finally, we obtain a UV texture with the duplicated face removed and place it with the hair color computed (see Figure 3.31 and 3.32).

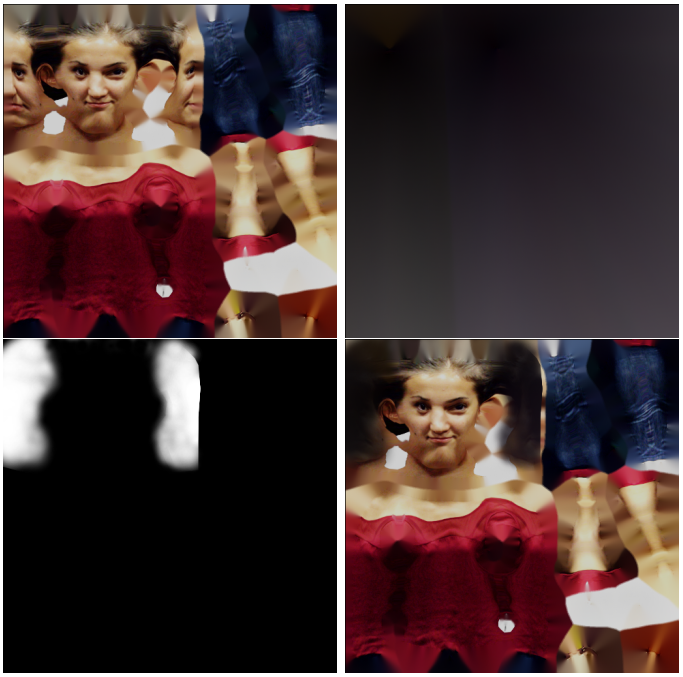


FIGURE 3.31: Initial baked image with empty areas filled (top left), image with the color hair extrapolated (top right), mask considering the hair area (bottom left) and the resulting image by compositing the initial baked image and the color hair image with the mask (bottom right).

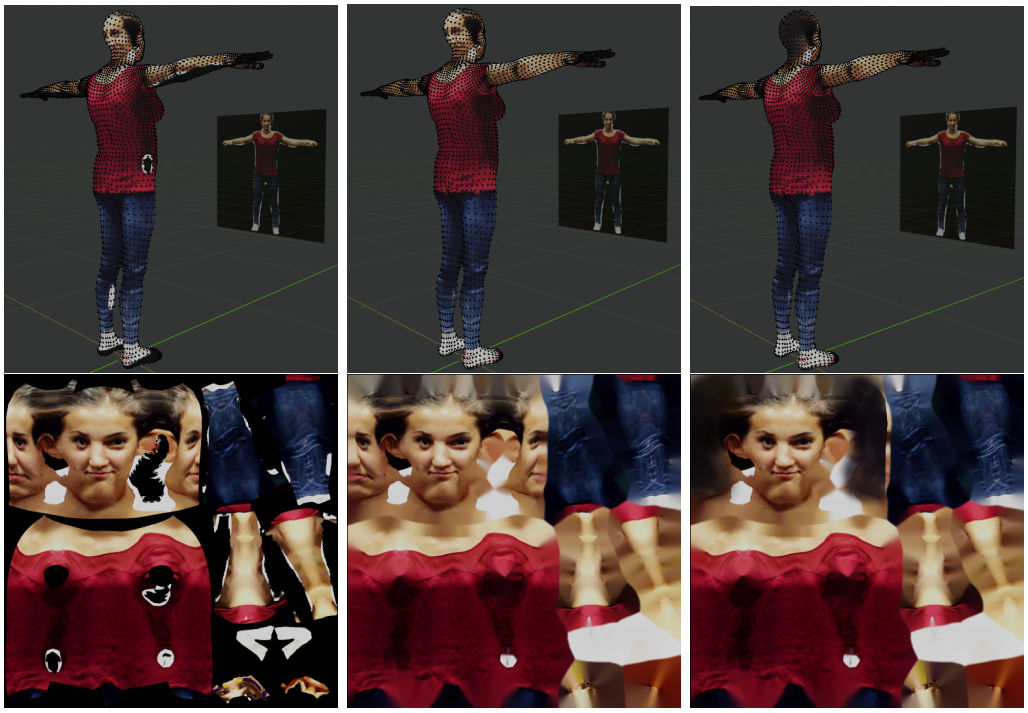


FIGURE 3.32: Avatar with the initial baked textured (left), avatar with the empty areas filled in the texture (middle) and avatar with the duplicated face removed (right).

Chapter 4

Overall pipeline

The input of our pipeline (see Figure 4.1) is a single image of a person (front view). We then compute the pose and face keypoints in the image with *OpenPose* and the silhouette of the 2D body with *Deeplab* network. We combine both information to segment the person's body with the head size as reference. We focus on the belly section of the person and calculate distance points from silhouette to the central point. We compare this distances with a database of templates and select the body shape that has a closer value distances through a correlation matrix.

After getting the closest mesh that fits better to the person's body shape. We project the initial image onto the mesh but before baking it on the object, we first adjust the pose of the avatar with the person's pose in the image.

After baking the image, we obtain a UV texture but there are some wrong areas we have to correct. First we fill the missing areas (in black) and then we remove the duplicated face (since the projection was done in both front and back directions to estimate texture colors on the back). Finally, we end up with a textured 3D full body avatar ready for animate.

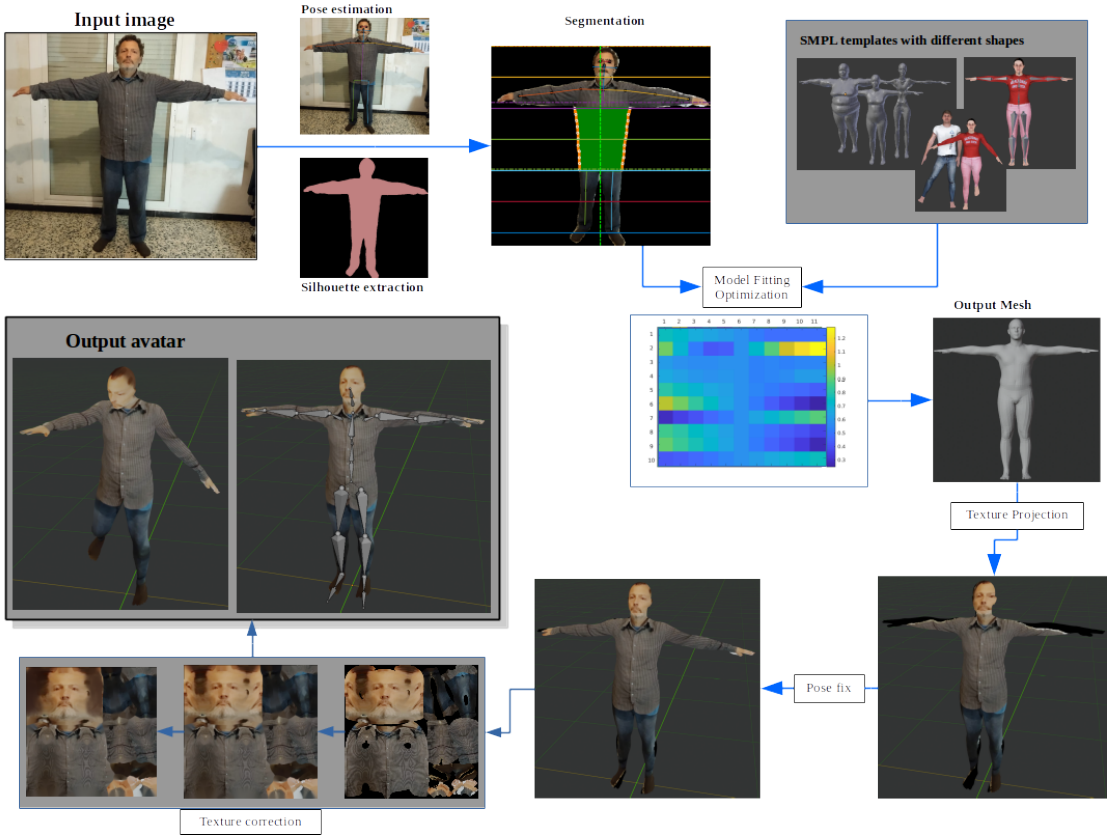


FIGURE 4.1: Our pipeline that computes a 3D full body avatar from a single image.

Chapter 5

Results

Our pipeline computes a 3D avatar from a single picture automatically in less than a minute. Each part of the method has a computation time of seconds (see Table 5.1). We did the computations in a laptop with processor Intel Core i5-4200M CPU @ 2.50GHz 4 and graphic card GeForce GTX 765M.

We run the method with the female and male templates to see how reliable is on computing the final 3D avatar. We compare the resulting mesh and texture with the original ones (see Figure 5.1 and 5.2). We can observe that the output mesh has similar shape than the original and the UV textures computed are close to the original ones. Although there are differences between the input and output avatar, the pipeline seems to achieve the goal fairly good. More results are shown in the end of this sections (see Figure 5.3).

Pipeline	Computation time (s)
Pose and face key points estimation	2.45 (x2)
Silhouette extraction	0.95
Image segmentation	9.52
Model fitting optimization	4.09
Texture projection	2.28
Pose fix + texture projection	2.32
pose + projection + bake	3.01
Texture projection+ pose fix + bake + texture correction	4.33
TOTAL	**23,79**

TABLE 5.1: Computation time of each part of the pipeline.

FIGURE 5.1: Pipeline applied on the initial female *SMPL* avatar.

FIGURE 5.2: Pipeline applied on the initial male *SMPL* avatar.

FIGURE 5.3: Some results of our pipeline.

Chapter 6

Conclusion and Future work

Although we only considered the belly part of the person's silhouette to compute the shape in 3D, the final virtual avatar appearance looks similar to the initial picture. This is achieved mostly due to the image projection and the correction step because the texture improves the identification of the self.

We could improve the final body shape taking into account more body parts to match the silhouette of the person. Regarding the texture, there are some parts that could be improved such as the hands or some wrong colors that had an incorrect estimation. We could separate each area of the mesh in the UV texture (legs, arms, foot, hands) and make a more deep correction taking each part separately in order to achieve an improved UV texture avoiding mismatch of colors between body parts.

References

[1] *https://avatarsdk.com*.

[2] *https://www.reallusion.com/character-creator/headshot/*.

[3] T. Alldieck, G. Pons-Moll, C. Theobalt, and M. Magnor, *Tex2shape: Detailed full human body geometry from a single image*, in IEEE International Conference on Computer Vision (ICCV), IEEE, 2019.

[4] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele, *2d human pose estimation: New benchmark and state of the art analysis*, in IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[5] M. Bertalmío, A. L. Bertozzi, and G. Sapiro, *Navier-stokes, fluid dynamics, and image and video inpainting*, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 1 (2001), pp. I–I.

[6] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, *Keep it SMPL: Automatic estimation of 3D human pose and shape from a single image*, in Computer Vision – ECCV 2016, Lecture Notes in Computer Science, Springer International Publishing, Oct. 2016.

[7] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh, *Openpose: Realtime multi-person 2d pose estimation using part affinity fields*, IEEE Transactions on Pattern Analysis and Machine Intelligence, (2019).

[8] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, *Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs*, CoRR, abs/1606.00915 (2016).

[9] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, *Encoder-decoder with atrous separable convolution for semantic image segmentation*, in ECCV, 2018.

[10] R. Drillis and R. Contini, *Body segment parameters*, vol. 3, Office of Vocational Rehabilitation. Department of health, Education and Welfare, New York, 1966.

[11] Y. Feng, F. Wu, X. Shao, Y. Wang, and X. Zhou, *Joint 3d face reconstruction and dense alignment with position map regression network*, in ECCV, 2018.

[12] R. A. Güler, N. Neverova, and I. Kokkinos, *Densepose: Dense human pose estimation in the wild*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 7297–7306.

[13] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft coco: Common objects in context*, in Computer Vision – ECCV 2014, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds., Cham, 2014, Springer International Publishing, pp. 740–755.

[14] M. Loper, N. Mahmood, J. Romero, G. Pons-Moll, and M. J. Black, *SMPL: A skinned multi-person linear model*, ACM Trans. Graphics (Proc. SIGGRAPH Asia), 34 (2015), pp. 248:1–248:16.

[15] G. Pavlakos, V. Choutas, N. Ghorbani, T. Bolkart, A. A. A. Osman, D. Tzionas, and M. J. Black, *Expressive body capture: 3d hands, face, and body from a single image*, in Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), 2019.

[16] J. Richards, *The Comprehensive Textbook of Clinical Biomechanics*, 03 2018.

[17] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, *Hand keypoint detection in single images using multiview bootstrapping*, in CVPR, 2017.

[18] M. Slater, S. Neyret, T. Johnston, G. Iruretagoyena, M. Álvarez de la Campa, M. Alabèrnia-Segura, B. Spanlang, and G. Feixas, *An experimental study of a virtual reality counselling paradigm using embodied self-dialogue*, 9 (2019).

[19] A. Telea, *An image inpainting technique based on the fast marching method*, J. Graphics, GPU, Game Tools, 9 (2004), pp. 23–34.